HPC on OpenStack

Review of the our Cloud Platform Project

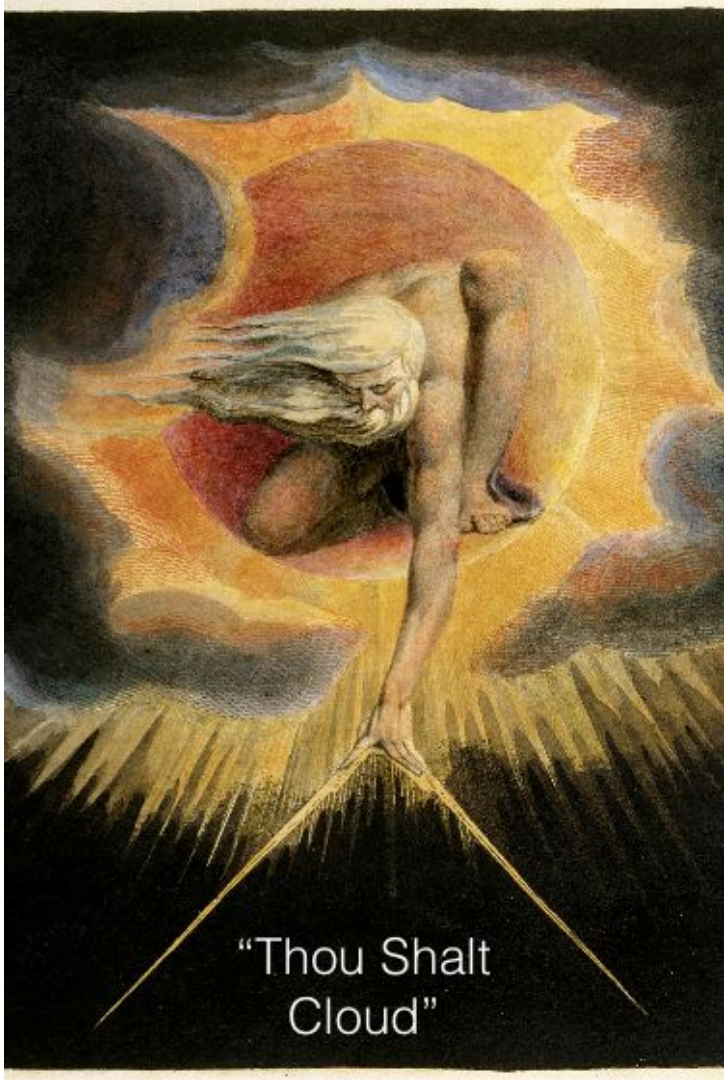
Petar Forai, Erich Bingruber, Uemit Seren
Post FOSDEM tech talk event @UGent 04.02.2019

Agenda

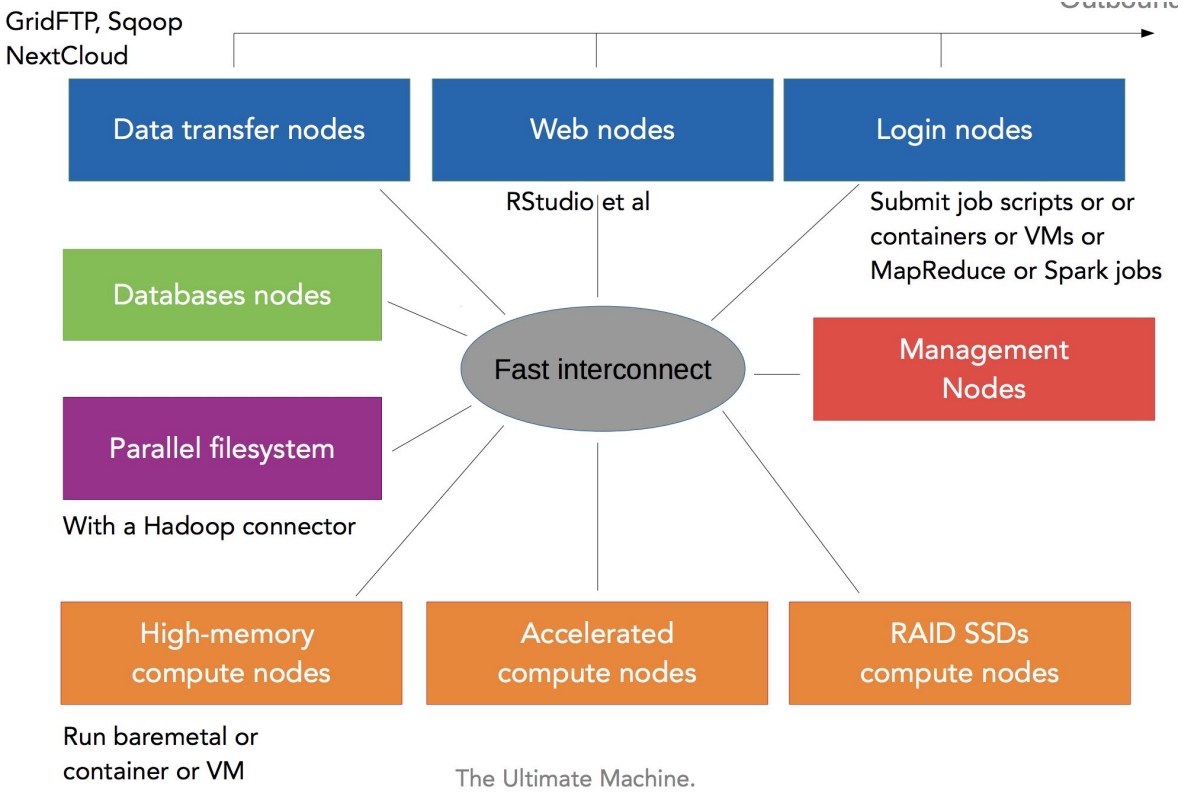
Who We Are & General Remarks (Petar Forai)

Cloud Deployment and Continuous Verification (Uemit Seren)

Cloud Monitoring System Architecture (Erich Birngruber)



The “Cloudster” and How we’re Building it!



Shamelessly stolen from Damien François Talk -- “*The convergence of HPC and BigData*
What does it mean for HPC sysadmins?”

Who Are We

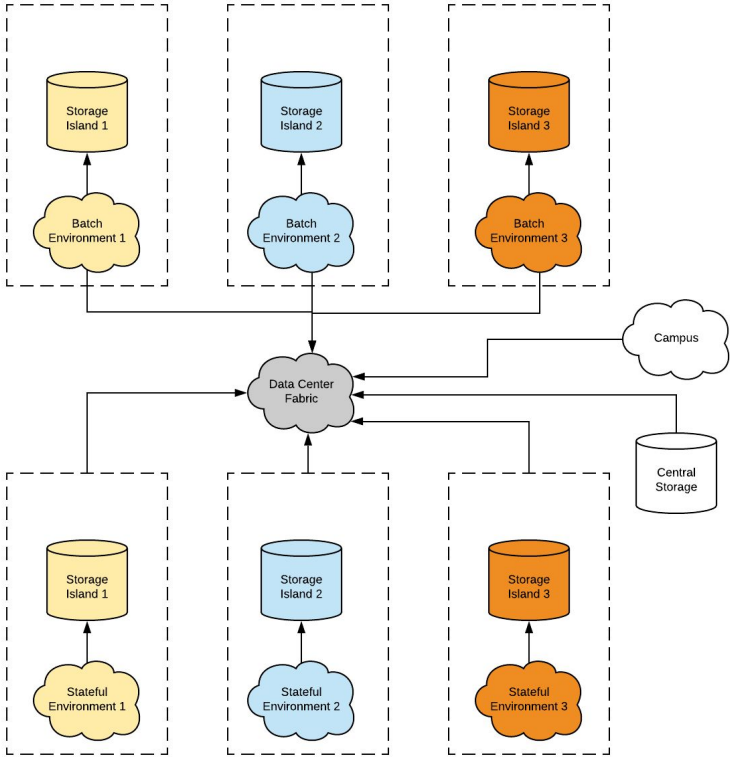
- Part of Cloud Platform Engineering Team at molecular biology research institutes (IMP, IMBA,GMI) located in Vienna, Austria at the Vienna Bio Center.
- Tasked with delivery and operations of IT infrastructure for ~ 40 research groups (~ 500 scientists).
- IT department delivers full stack of services from workstations, networking, application hosting and development (among many others).
- Part of IT infrastructure is delivery of HPC services for our campus
- 14 People in total for everything.

Vienna Bio Center Computing Profile

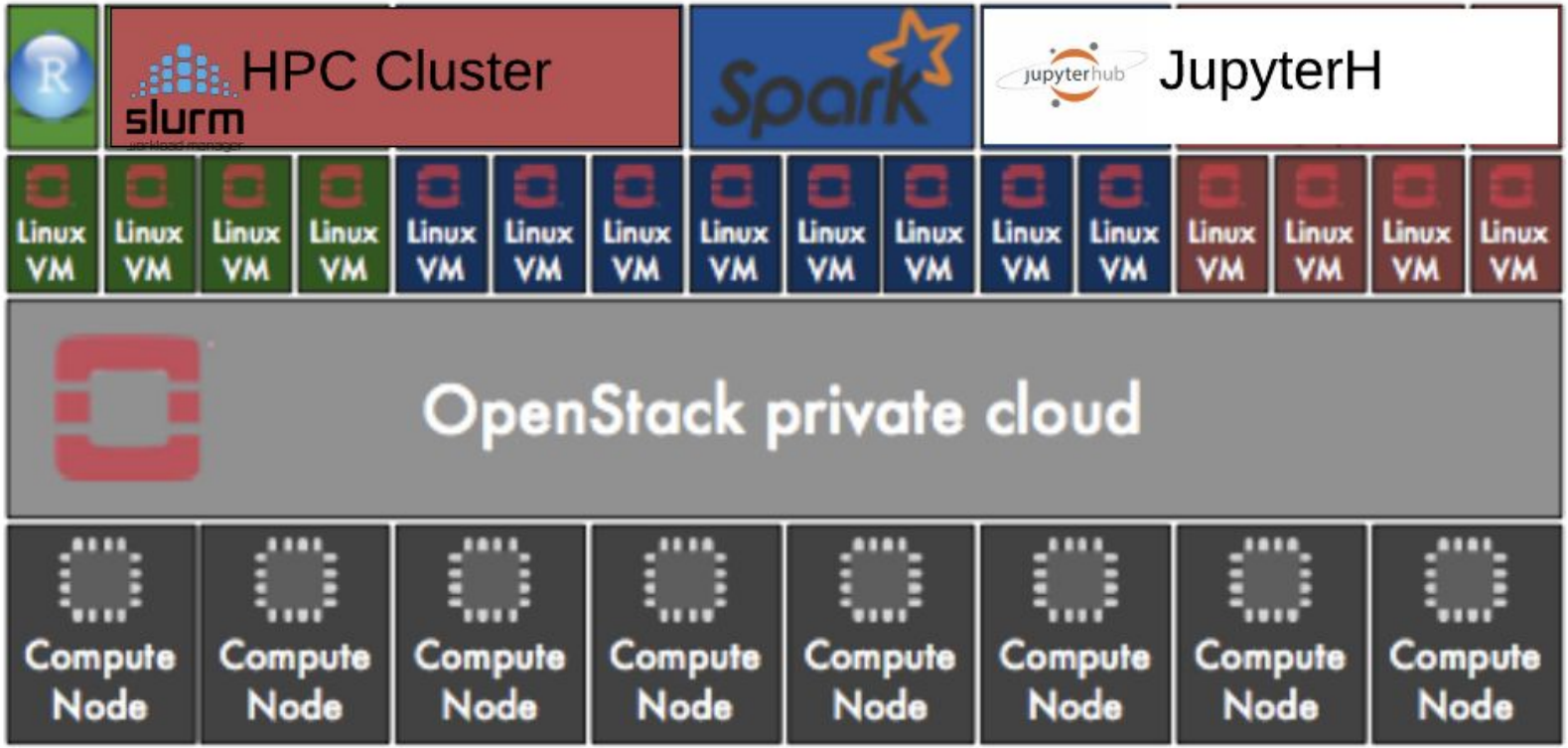
- Computing infrastructure almost exclusively dedicated to bioinformatics (genomics, image processing, cryo electron microscopy, etc.)
- Almost all applications are data exploration, analysis and data processing, no simulation workloads
- Have all machinery for data acquisition on site (sequencers, microscopes, etc.)
- Operating and running several compute clusters for batch computing and several compute clusters for stateful applications (web apps, data bases, etc.)

What We Currently Have

- Siloed islands of infrastructure
- Cant talk to other islands, can’t access data from other island (or difficult logistics for users)
- Nightmare to manage
- No central automation across all resources easily possible



What We're Aiming At



Meet the CLIP Project

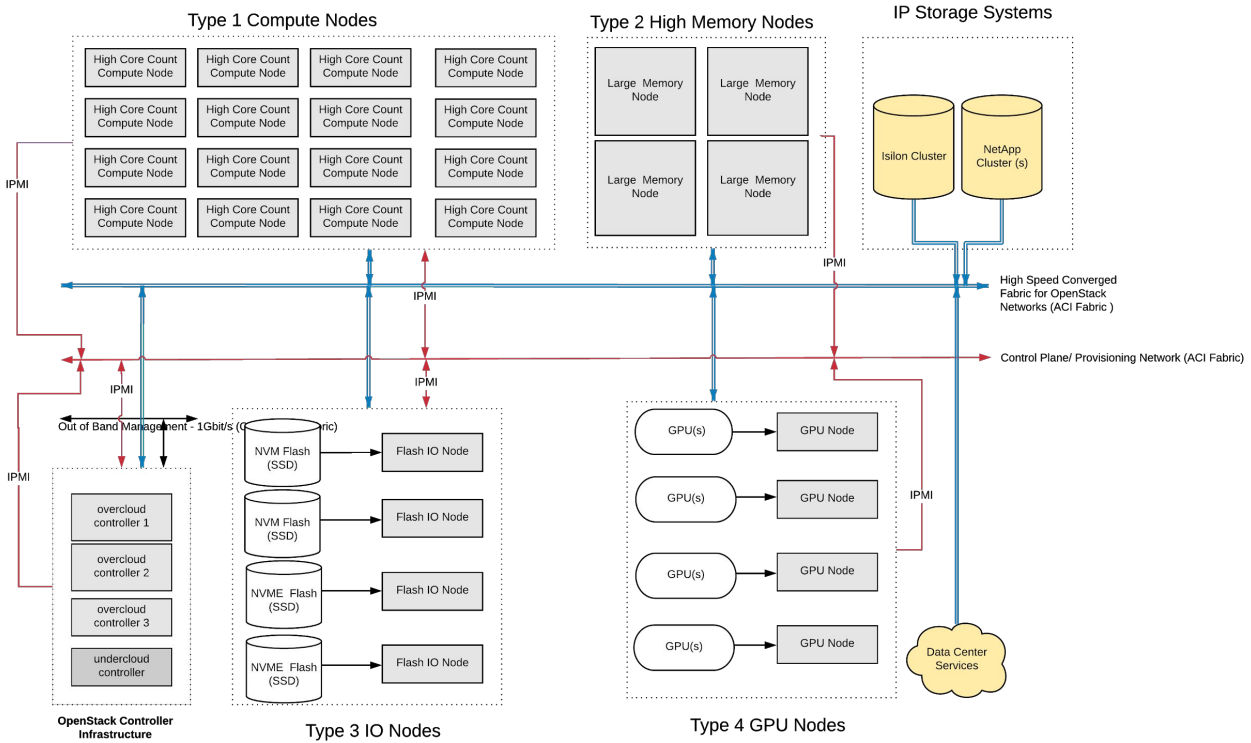
- OpenStack was chosen to be evaluated further as platform for this
- Setup a project “CLIP” (Cloud Infrastructure Project) and formed project team (4.0 FTE) with a multi phase approach to delivery of the project.
- Goal is to implement not only a new HPC platform but a software defined datacenter strategy based on OpenStack and deliver HPC services on top of this platform
- Delivered in multiple phases

Tasks Performed within “CLIP”

- Build PoC environment to explore and develop understanding of OpenStack (~ 2 months)
- Start deeper analysis of how OpenStack (~ 8 months)
  - Define and develop architecture of the cloud (understand HPC specific impact)
  - Develop deployment strategy and pick tooling for installation, configuration management, monitoring, testing
  - Develop integration into existing data center resources and services
  - Develop understanding for operational topics like development procedures, upgrades, etc.
  - Benchmark

- Deploy production Cloud (~ 2 months and ongoing)
  - Purchase and install hardware
  - Develop architecture and pick tooling for for payload (HPC environments and applications)
  - Payload deployment

CLIP Cloud Architecture Hardware



- Heterogeneous nodes (high core count, high clock, high memory, GPU accelerated, NVME)
- First phase ~ 100 compute nodes and ~ 3500 Intel SkyLake cores
- 100GbE SDN RDMA capable Ethernet and some nodes with 2x or 4x ports
- ~ 250TB NVMe IO Nodes ~ 200Gbyte/s

HPC Specific Adaptations

- Tuning, Tuning, Tuning required for excellent performance
  - NUMA clean instances (KVM process layout)
  - Huge pages (KSM etc.) setup
  - Core isolation
  - PCI-E passthrough (GPUs, NVME, …) and SR-IOV (esp. for networking) crucial for good performance

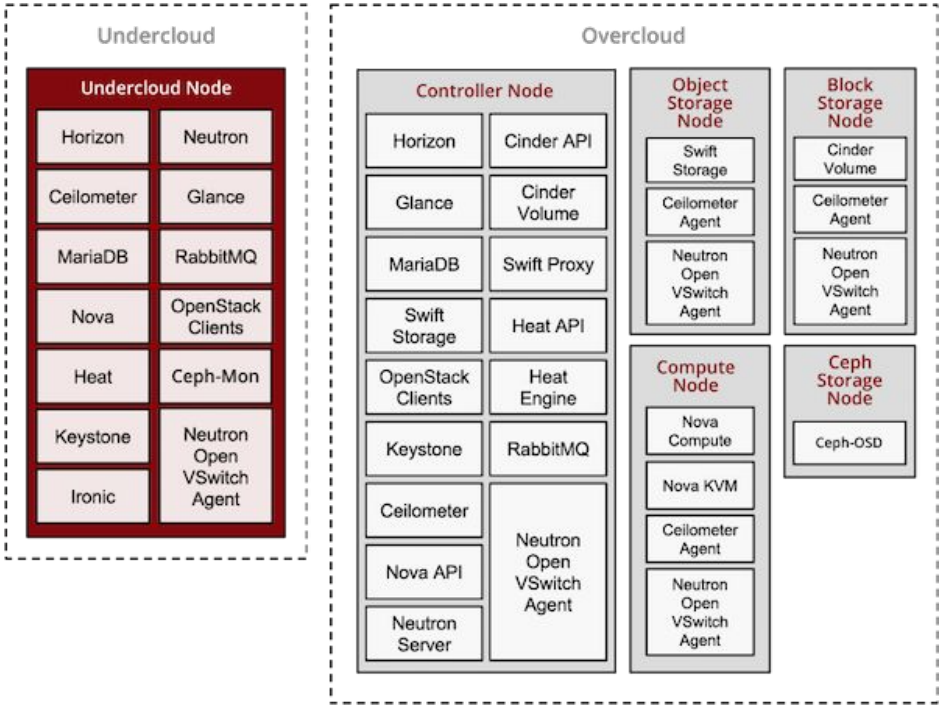
Lessons Learned

- OpenStack is *incredibly* complex
- OpenStack is not a product. It is a framework.
- You need 2-3 OpenStack environments (development, staging, prod in our case) to practice and understand upgrades and updates.
- Out of box experience and scalability for certain OpenStack subcomponents is not optimal and should be considered more of a reference implementation
    - Consider plugging in real hardware here
- Cloud networking is really hard (especially in our case)

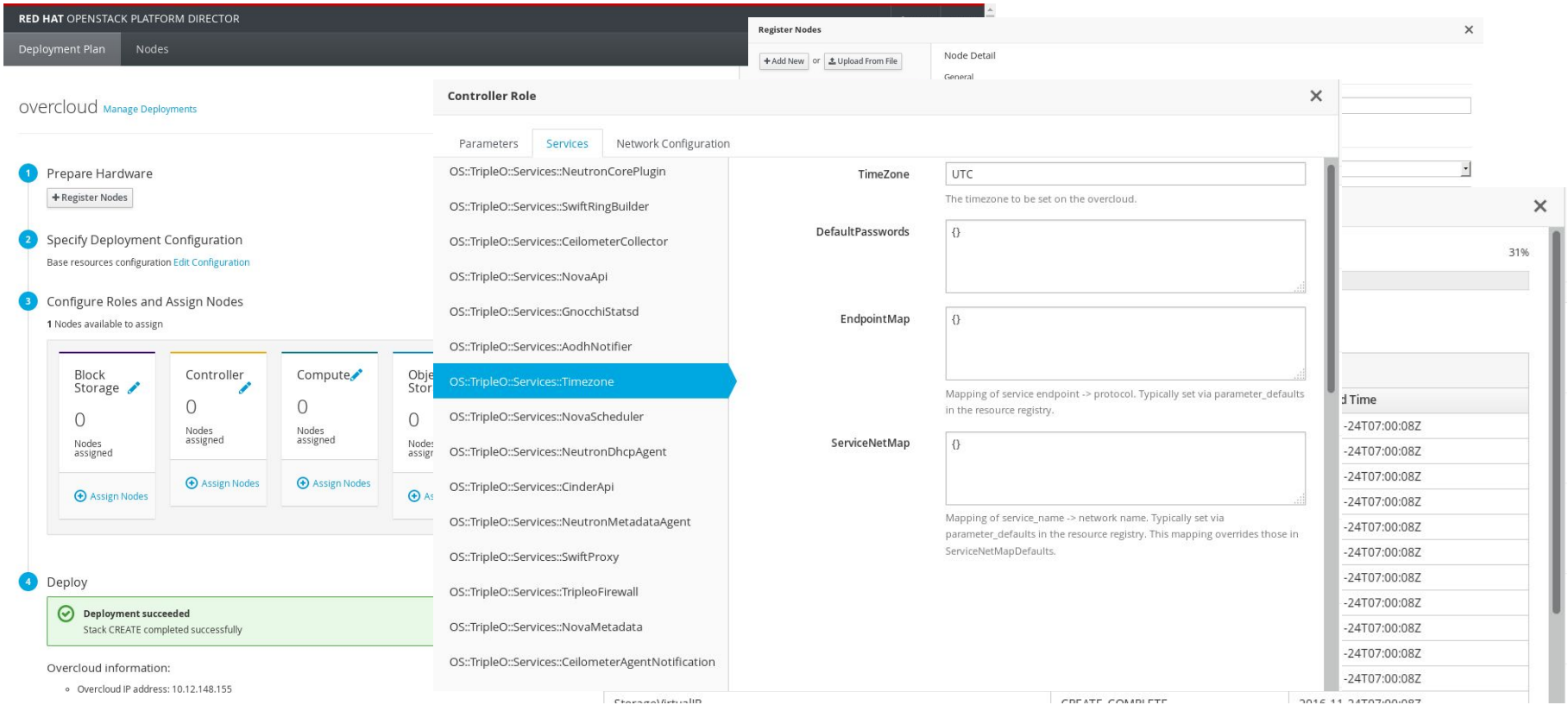
Deployment and Cloud Verification

Deployment and Cloud Verification

- Red Hat OpenStack (OSP) uses the upstream “TripleO” (OpenStack on OpenStack) project for the OpenStack deployment.
- Undercloud (Red Hat terminology: Director) is a single node deployment of OS using puppet.
- The Undercloud uses various OpenStack projects to deploy the Overcloud which is our actual cloud where payload will run
- TripleO supports deploying a HA overcloud
- Overcloud can be installed either using a Web-GUI or entirely from the CLI by customizing yaml files.



Deployment and Cloud Verification



Deployment and Cloud Verification

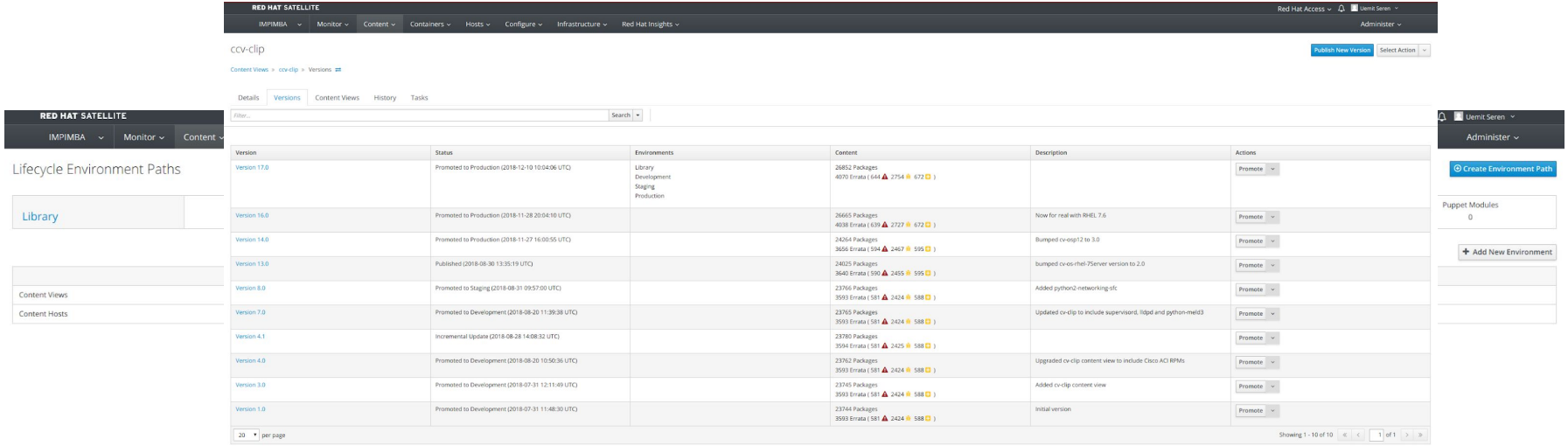
- Web GUI is handy to play around but not so great for fast iterations and Infra as code.
  → **Disable the Web UI and deploy from the CLI.**
- TripleO internally uses ***heat*** to drive ***puppet*** that drives ***ansible*** ¯_(ツ)_/¯
- We decided to use ***ansible*** to drive the TripleO OpenStack deployment.
- Deployment split in 4 phases corresponding to 4 git repos:
  a. *clip-undercloud-prepare*: Ansible playbooks that run on a bastion VM to prepare and install the undercloud using PXE and kickstart.
  b. *clip-tripleo* contains the customized yaml files for the TripleO configuration (storage, network settings, etc)
  c. *clip-bootstrap* contains ansible playbooks to initially deploy or update the overcloud using the configuration in the clip-tripleo repo
  d. *clip-os-infra* contains post deployment customizations that are not exposed through TripleO or very cumbersome to customize

Deployment and Cloud Verification

- TripleO is slow because Heat → Puppet → Ansible !!
  - Small changes require “stack update” → 20 minutes (even for simple config stanza change and service restart).
- Why not move all customizations to ansible (clip-os-infra) ? Unfortunately not robust :-(
  - Stack update (scale down/up) will overwrite our changes
  - → services can be down
- Let’s compromise:
  - Iterate on different customizations using ansible
  - Move finalized changes back to *TripleO*
- Ansible everywhere else !
  - *clip-aci-infra*: prepare the networking primitives for the 3 different OpenStack environments
  - Move nodes between environments in the network fabric

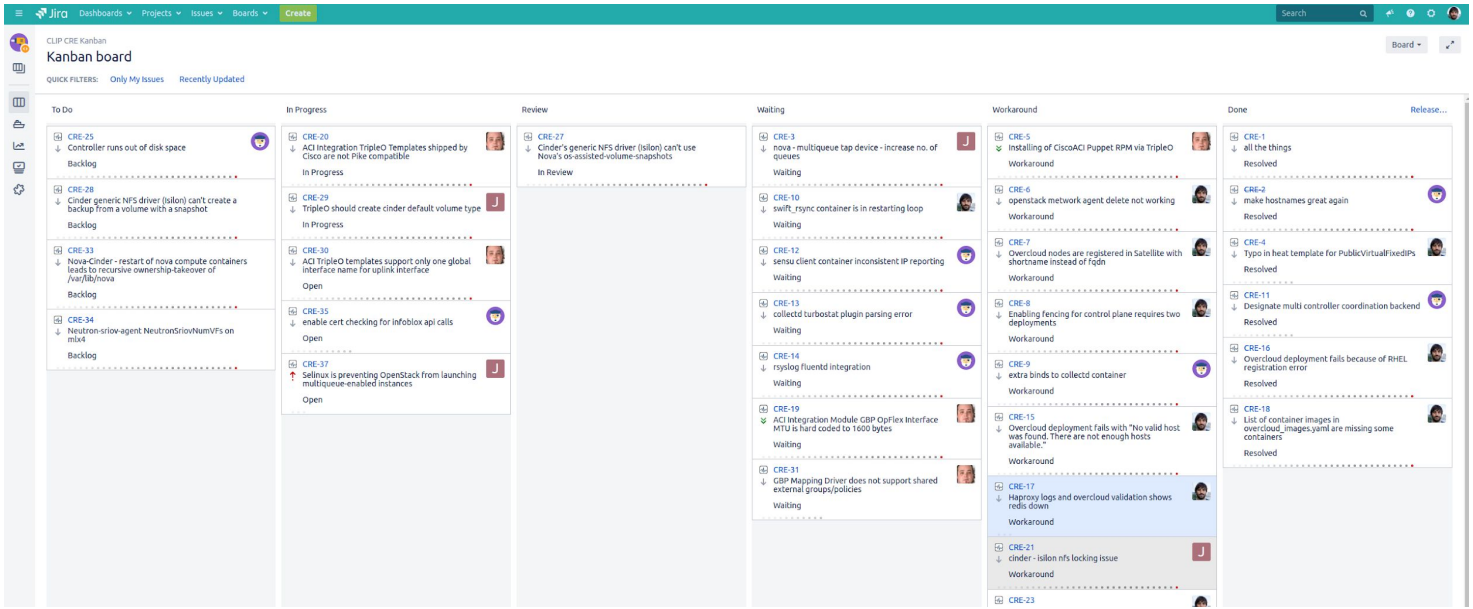
Deployment and Cloud Verification

- We have 3 different environments (dev, staging and production) to try out updates and configuration changes. We can guarantee reproducibility of deployment because we have everything as code/yaml, but what about software packages ?
- To make sure that we can predictably upgrade and downgrade we decided to use Red Hat Satellite (Foreman) and create Content Views and Life Cycle Environments for our 3 environments



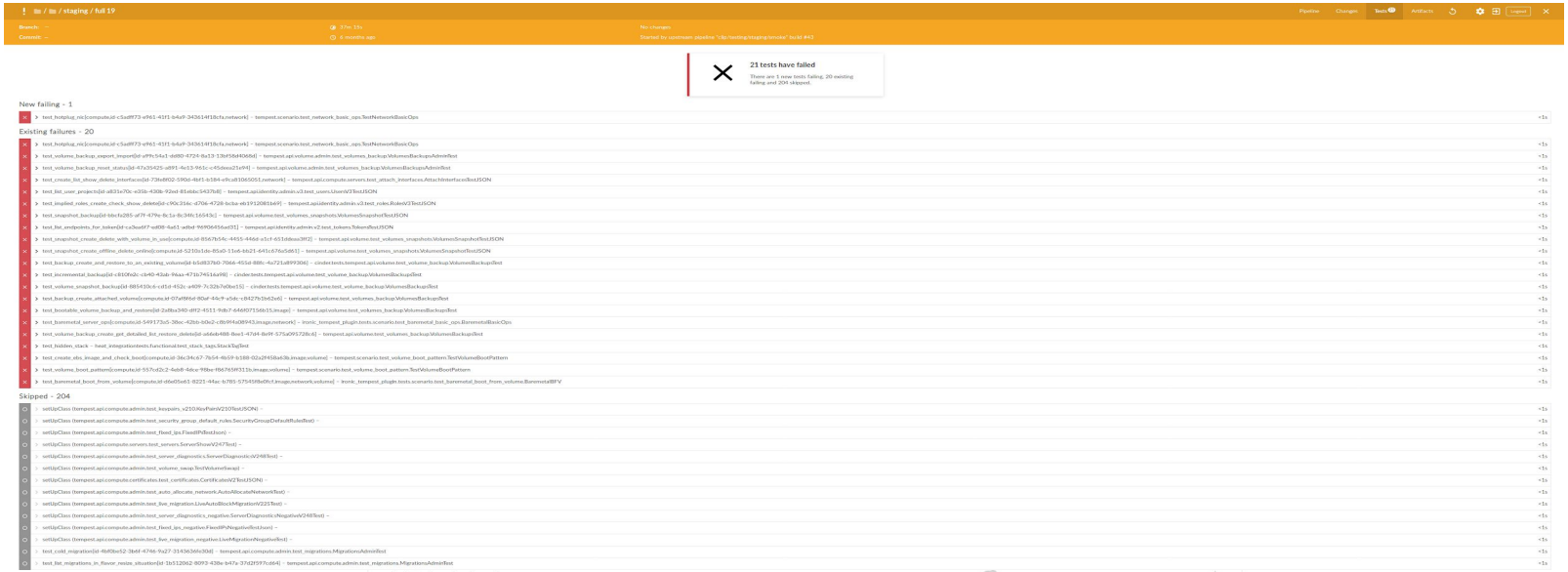
Deployment and Cloud Verification

- While working on the deployment we ran into various known bugs that are fixed in newer versions of OSP. To keep track of the workaround and the status of those bugs we use a dedicated JIRA project (CRE)



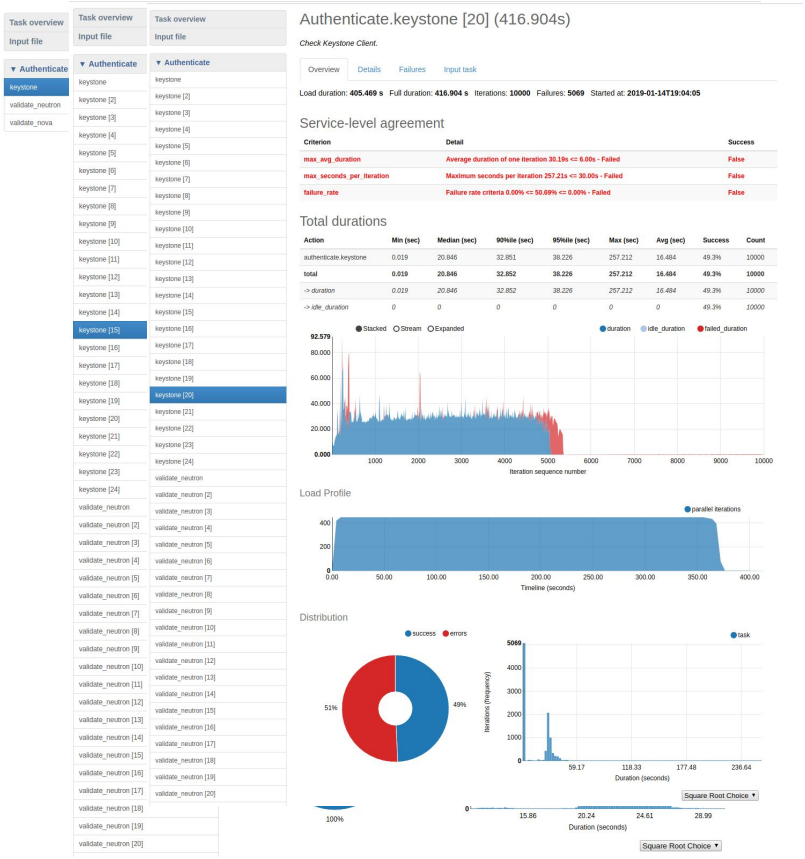
Deployment and Cloud Verification

- How can we make sure and monitor that the cloud works during operations ?
- We leverage OpenStack's own tempest testing suite to run verification against our deployed cloud.
- First smoke test (~ 128 tests) and if this is successful run full test (~ 3000 tests) against the cloud.



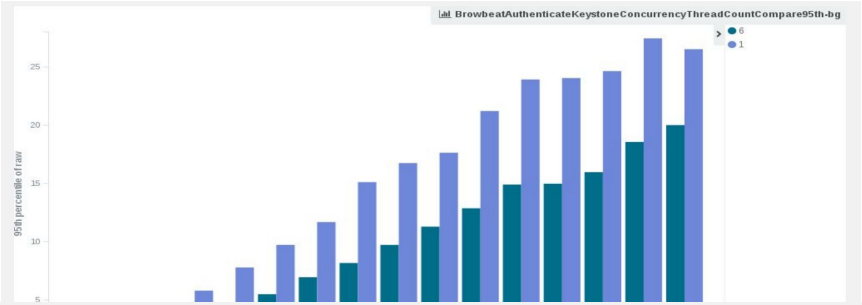
Deployment and Cloud Verification

- Ok, the Cloud works but what about performance ? How can we make sure that OS performs when upgrading software packages etc ?
- We plan to use *Browbeat* to run *Rally* (control plane performance/stress testing), *Shaker* (network stress test) and *PerfkitBenchmarker* (payload performance) tests on a regular basis or before and after software upgrades or configuration changes



Deployment and cloud verification

- Grafana and Kibana dashboard can show more than individual rally graphs:



- Browbeat can show differences between settings or software versions:

```
Scrolling through Browbeat 22 documents...
+-------------------------------------------------------------------------------------+
Scenario           | Action                | conc.| times | 0b5ba58c | 2b177f3b  | % Diff
+-------------------------------------------------------------------------------------+
create-list-router | neutron.create_router | 500  |   32  | 19.940   | 15.656    | -21.483
create-list-router | neutron.list_routers  | 500  |   32  |  2.588   |  2.086    | -19.410
create-list-router | neutron.create_network| 500  |   32  |  3.294   |  2.366    | -28.177
create-list-router | neutron.create_subnet | 500  |   32  |  4.282   |  2.866    | -33.075
create-list-port   | neutron.list_ports    | 500  |   32  | 52.627   | 43.448    | -17.442
create-list-port   | neutron.create_network| 500  |   32  |  4.025   |  2.771    | -31.165
create-list-port   | neutron.create_port   | 500  |   32  | 19.458   |  5.412    | -72.189
create-list-subnet | neutron.create_subnet | 500  |   32  | 11.366   |  4.809    | -57.689
create-list-subnet | neutron.create_network| 500  |   32  |  6.432   |  4.286    | -33.368
create-list-subnet | neutron.list_subnets  | 500  |   32  | 10.627   |  7.522    | -29.221
create-list-network| neutron.list_networks | 500  |   32  | 15.154   | 13.073    | -13.736
create-list-network| neutron.create_network| 500  |   32  | 10.200   |  6.595    | -35.347
+-------------------------------------------------------------------------------------+
+-------------------------------------------------------------------------------------+
UUID                                  | Version |    Build      | Number of runs
+-------------------------------------------------------------------------------------+
938dc451-d881-4f28-a6cb-ad502b177f3b  | queens  | 2018-03-20.2  |       1
6b50b6f7-acae-445a-ac53-78200b5ba58c  | ocata   | 2017-XX-XX.X  |       3
+-------------------------------------------------------------------------------------+
```

Deployment and cloud verification

Lessons learned and pitfalls of OpenStack/Tripleo:

- OpenStack and TripleO are complex with many moving parts. → **Have a dev/staging environment to test the upgrade and pin the software versions with Satellite or Foreman.**
- Upgrades (even minor ones) can break the cloud in unexpected ways. Biggest pain point was to upgrade from OSP11 (non-containerized) -> OSP12 (containerized).
- Containers are no free lunch. You need a container build pipeline to customize upstream containers to add fixes and workarounds.
- TripleO gives you a supported out of the box installer for HA OpenStack with common customizations. Non-common customizations are hard because of rigid architecture (heat, puppet, ansible mixed together). TripleO is moving more towards ansible (config download)
- "*Flying blind through clouds is dangerous*": Make sure you have a pipeline for verification and performance regression testing.
- Infra as code (end to end) is great but requires discipline (proper PR reviews) and release management for tracking workarounds and fixes.

Cloud Monitoring System Architecture

Monitoring is Difficult

Because it’s hard to get these right

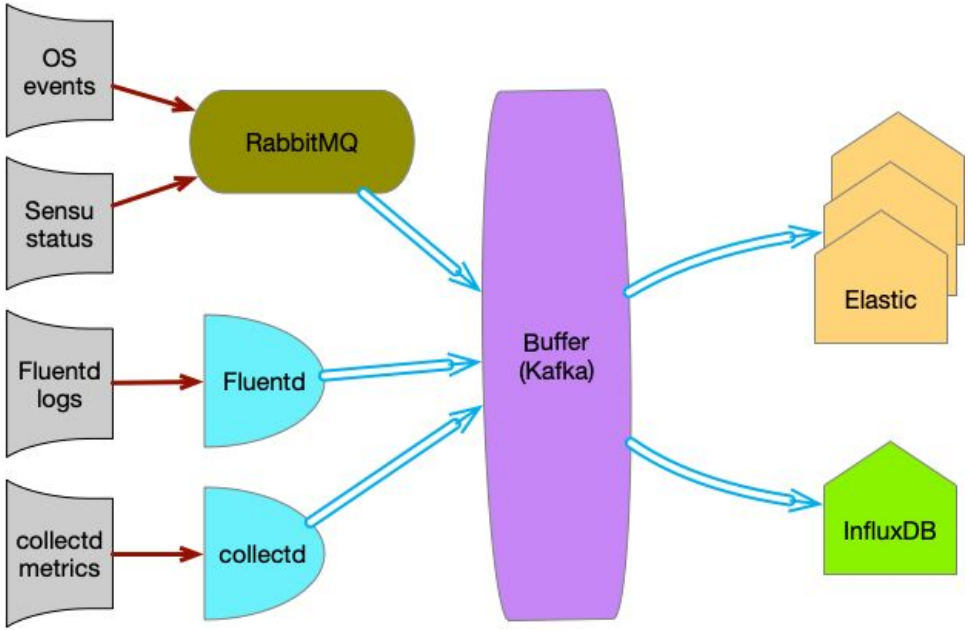
- The information
  - At the right time
  - For the right people
- The numbers
  - Too few alarms
  - Too many alarms
  - Too many monitoring systems
- The time
  - doing it too late

Monitoring: What We Want to Know

- Logs: as structured as possible → Fluentd
  - syslog (unstructured)
  - OpenStack logs (structured)
- Events → RabbitMQ
  - OpenStack RPCs
  - high-level OpenStack interactions
  - CRUD of resources
- Status → Sensu
  - polling: is the service UP?
  - Publish / subscribe
  - modelling service dependencies
- Metrics → Collectd
  - time series, multi dimensional
  - performance metrics

Monitoring: How We do it

- Architecture
  - Ingest endpoints for all protocols
  - Buffer for peak loads
  - Persistent store
    - Structured data
    - timeseries
- Dashboards
  - Kibana, Grafana
  - Alerta to unify alarms
- Integration with deployment
  - Automatic configuration
- Service catalog integration
  - Service owners
  - Pointers to documentation



Monitoring: Outlook

- What changes for cloud deployments
  - Lifecycle, services come and go
  - Services scale up and down
  - No more hosts

- Further improvements
  - infrastructure debugger (tracing)
  - Stream processing (improved log parsing)
  - Dynamically integrate call-duty / notifications / handover
  - Robustness (last resort deployment)

Thanks!